In ISHEM 2005: the First International Forum on Integrated System Health Engineering and Management in Aerospace, Napa, California, Nov. 7-10, 2005

V&V of ISHM for Space Exploration

Lawrence Markosian, QSS Group, Inc./NASA Ames Research Center
Martin S. Feather, Jet Propulsion Laboratory, California Institute of Technology
David Brinza, Jet Propulsion Laboratory, California Institute of Technology
Fernando Figueroa, NASA Stennis Space Center

NASA has established a far-reaching and long-term program for robotic and manned exploration of the solar system, beginning with missions to the moon and Mars. Integrated System Health Management (ISHM) will be key to improving the reliability, operability and maintainability of many of the systems deployed in this endeavor. In this setting, Verification and Validation (V&V) and certification of ISHM systems will be necessary but challenging.

The factors that most influence ISHM's V&V and certification needs stem from two main sources – the system of which ISHM is a part, and the implementation of ISHM itself. The system of which ISHM is a part levies requirements on ISHM – for example, the need for ISHM to respond within a given time period with a stipulated level of confidence in the correctness of its response. The combination of these *externally* imposed requirements, coupled with the manner in which ISHM will be utilized, drive much of the V&V and certification process. Also highly influential is the nature of the ISHM implementation. Often it takes a combination of techniques to implement an ISHM system. These techniques include well-understood algorithms for low-level data analysis, validation and reporting; traditional capabilities for fault detection, isolation and recovery; and, at the more novel end, Artificial Intelligence (AI) techniques for state estimation and planning. Detailed descriptions of these techniques are beyond the scope of this paper, and may be found elsewhere. Here we focus on their ramifications for V&V and certification. In particular, this range of techniques that will be utilized within an overall ISHM system impose *internal* challenges to V&V.

The conjunction of these external and internal influences on ISHM V&V and certification, and the challenges that stem from them, is the focus of this paper. We outline existing V&V approaches and analogs in other software application areas, and possible new approaches to the V&V challenges for space exploration ISHM.

1 EXISTING SOFTWARE V&V AND CERTIFICATION PRACTICES

There are many areas other than space flight where embedded systems exhibit a safety critical role, for example commercial aircraft avionics, defense, medical devices, nuclear power, and transportation. In this section we begin by looking at existing V&V and certification practices for ISHM as seen in one of these areas, commercial aircraft avionics. This area has many parallels to the safety- and mission-critical needs that predominate in NASA's space activities. We then suggest that the existing NASA hierarchy of requirements, policies, standards and procedures relevant to software has close parallels with those seen in the other safety critical areas.

1.1 Avionics V&V and Certification

Safety-critical software for commercial aircraft undergoes certification by the FAA, which includes V&V in accordance with RTCA/DO-178B. This document is recognized as the means for evaluating software for compliance with the relevant Federal Aviation Regulations/Joint Aviation Regulations (FARs/JARs) for embedded systems in commercial aircraft. A useful paper [Johnson, 1998] provides interpretation of RTCA/DO-178B; it was prepared by a Boeing participant in the RTCA committee responsible for DO-178B. The paper describes the intent and rationale of DO-178B. The derivation of the software approval guidelines from the Federal Aviation Regulations (FARs) to DO-178B is discussed in the paper to clarify its relationship to the government regulations. An explanation of the Designated Engineering Representative (DER) system is also provided in the paper along with a discussion of the safety process to describe the environment in which DO-178B is used.

The DO-178B/ED-12B Software Verification Process defines specific verification objectives that must be satisfied; these include:

a. Verification of software development processes,
b. Review of software development life cycle data,
c. Functional Verification of software
    i. Requirements-based testing and analysis
    ii. Robustness testing
d. Structural Coverage Analysis

Verification of the software development processes is accomplished by a combination of reviews and analyses. For software requirements, these include reviews of the quality of the requirements themselves, a requirements trace from system-level to low-level (code), and checks of their compatibility with the hardware; verifiability; conformance with standards; accuracy, correctness and behavior of algorithms. The software architecture is reviewed and analyzed for compatibility with the high-level requirements and target hardware. Conformance of the software architecture to standards, verifiability, consistency and portioning integrity is also reviewed. The source code is also subjected to compliance and traceability to requirements. Conformance of the source code to standards, code verifiability, accuracy and consistency are also reviewed and analyzed. The integration process is verified by examination of the data and memory maps (detect memory overlaps or missing components).

DO-178B section 11 stipulates a number of data requirements: plans, standards, procedures, and products (including the source code and executable code) that document this certification. These are:

Plan for Software Aspects of Certification
Software Development Plan
Software Verification Plan
Software Configuration Management Plan
Software Quality Assurance Plan
Software Requirements Standards
Software Design Standards
Software Code Standards
Software Requirements Data
Software Design Description

Source Code
Executable Object Code
Software Verification Cases and Procedures
Software Verification Results
Software Life Cycle Environment Configuration Index
Software Configuration Index
Problem Reports
Software Configuration Management Records
Software Quality Assurance Records
Software Accomplishment Summary

The review of software development life cycle data includes assessment of the test results, configuration management and quality assurance aspects for the development. The testing portion, due to its complexity, is described in detail below. The control of the configuration of the software, including identification of configuration items, establishment of configuration item baselines, change control data, and traceability throughout the development cycle is reviewed and analyzed. Problem reporting, tracking and corrective action records are reviewed for adequacy, and verification of a change is confirmed via examination of configuration records. The software quality assurance records are reviewed to provide confidence that the software life cycle processes have been followed and that deficiencies encountered in the life cycle are detected, evaluated, tracked and resolved.

Functional verification of the software is performed at three levels. (1) Hardware/software integration testing is performed to verify the correct operation of the software in the target computer environment. (2) Software integration testing verifies the interrelationships between software requirements and components and the implementation of the software components within the architecture. (3) Low-level testing verifies the implementation of software low-level requirements. These requirements-based tests are performed to verify correct functionality of the software in both normal range test cases and in robustness test cases. The normal range test cases utilize valid representative input values drawn from those normal input ranges (typically utilizing values at the range boundaries, and representative interior values), and use them to exercise the transitions possible in normal operation. The robustness test cases inject invalid input values, values that would generate arithmetic overflows or attempt to provoke transitions that are not allowed. The software should follow expected behavior for the abnormal cases.

Structural coverage analysis is generally perceived to be the most difficult task to undertake in the testing process. Furthermore, certifying real-time executable code with an operating system that is tightly integrated with the hardware, cache, interrupts, memory management, and process/task management, can make structural testing even more difficult. These low-level aspects create a significant challenge to the verification process. Three primary levels of structural testing are invoked according to the criticality level of the software (Table 2) in DO-178B certifications:

- Statement Coverage (SC): Every statement in the program has been invoked or used at least once. This is the most common use of the term “code coverage.”
- Decision Coverage (DC): Every point of entry and exit in the program has been invoked at least once and that each decision in the program has been taken on all possible (Boolean) outcomes at least once. Essentially, this means that every Boolean statement has been evaluated both TRUE and FALSE.
- Modified Condition Decision Coverage (MCDC): Every point of entry and exit in the program has been invoked at least once, that every decision in the program has taken all possible outcomes at least once, and that each condition in a decision has been shown to independently affect that decision's outcome. Complex Booleans need to have truth tables developed to set each variable (inside a Boolean expression) to both TRUE and FALSE.

In DO-178B terms, software has a criticality level, ranging from the most critical (“Level A”), down to “Level E”. “Level A” software requires *all three* levels of structural testing be performed.

Performing this code coverage exercise is possible using manual methods, but this process is now readily facilitated by utilizing commercial code coverage tools. Numerous code coverage tool vendors now supply testing tools that create the appropriate test outputs to demonstrate and satisfy compliance with DO-178B.

1.2 NASA Requirements, Policies, Standards and Procedures relevant to Software

The current NASA Software Safety Standard is NASA-STD-8719.13b, dated July 8, 2004, which applies to all safety-critical software acquired or produced by NASA. By reference this includes NASA Software Assurance Standard, NASA-STD-8739.8, dated July 28, 2004. This in turn includes by reference NASA Software Engineering Requirements, NPR 7150.2, Sept. 27, 2004. The latter characterizes “Class A Human Rated Software Systems” as:

> *Applies to all space flight software subsystems (ground and flight) developed and/or operated by or for NASA to support human activity in space and that interact with NASA human space flight systems. Space flight system design and associated risks to humans are evaluated over the program's life cycle, including design, development, fabrication, processing, maintenance, launch, recovery, and final disposal. Examples of Class A software for human rated space flight include but are not limited to: guidance; navigation and control; life support systems; crew escape; automated rendezvous and docking; failure detection, isolation and recovery; and mission operations.*

The classifications in NPR 7150.2 are important because, *inter alia*, the software engineering requirements, including V&V, depend on the classification. ISHM software is clearly Class A by this definition.

NPR 7150.2 also “provides a common set of generic requirements for software created and acquired by or for NASA...” Included in this document is a summary of the requirements with respect to software created and acquired by NASA. Figure 1, taken from this NPR, shows the relationships among the various relevant NASA requirements, policies, standards, procedures and guidance.

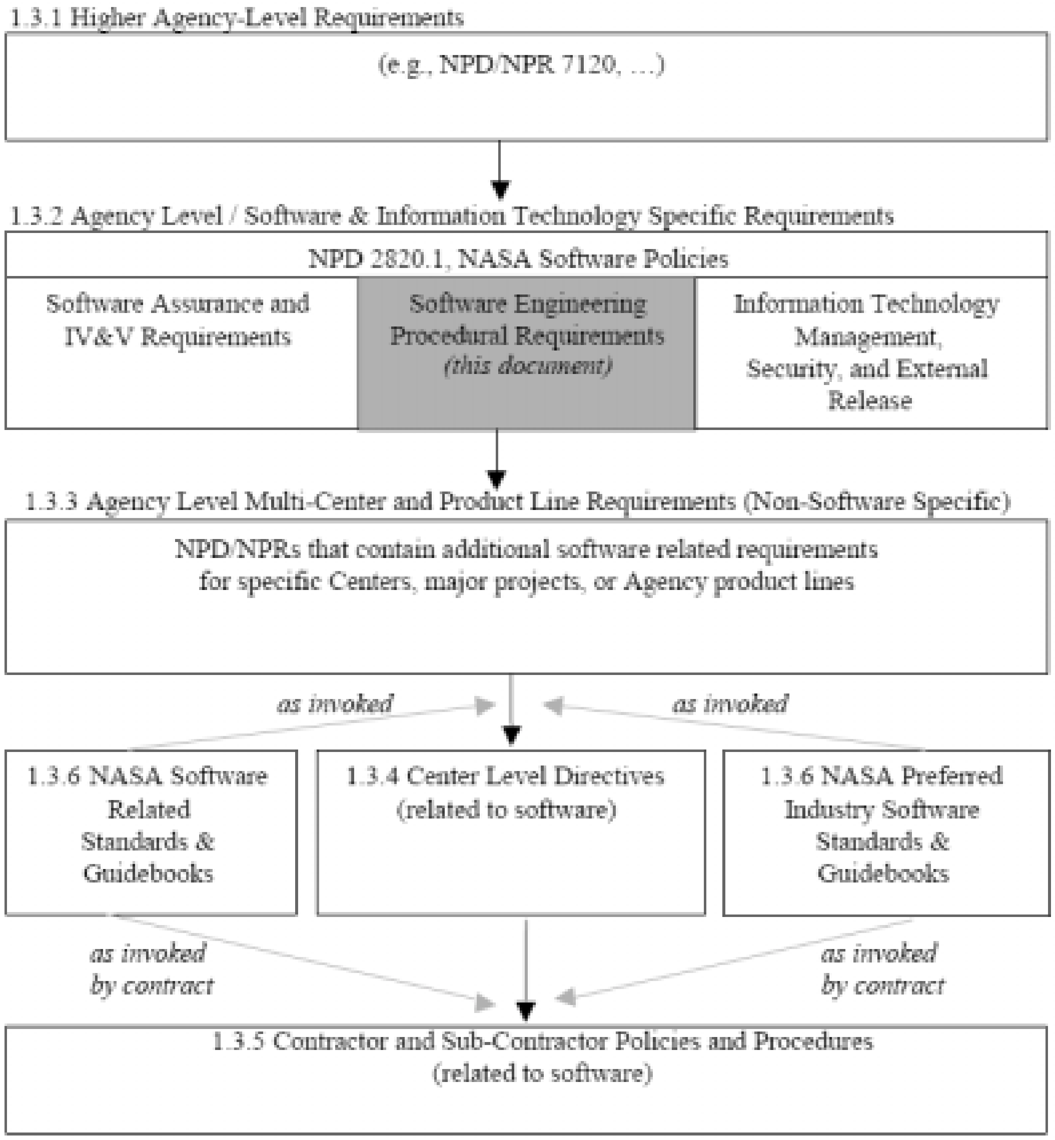
1.3.1 Higher Agency-Level Requirements
(e.g., NPD/NPR 7120, …)
1.3.2 Agency Level / Software & Information Technology Specific Requirements
NPD 2820.1, NASA Software Policies
Software Assurance and IV&V Requirements
Software Engineering Procedural Requirements (this document)
Information Technology Management, Security, and External Release
1.3.3 Agency Level Multi-Center and Product Line Requirements (Non-Software Specific)
NPD/NPRs that contain additional software related requirements for specific Centers, major projects, or Agency product lines
as invoked
as invoked
1.3.6 NASA Software Related Standards & Guidebooks
1.3.4 Center Level Directives (related to software)
1.3.6 NASA Preferred Industry Software Standards & Guidebooks
as invoked by contract
as invoked by contract
1.3.5 Contractor and Sub-Contractor Policies and Procedures (related to software)

Figure 1. Relationships among governing NASA software documents.

The net result of these governing documents is an approach to V&V and certification that has close parallels with those followed in other safety-critical application areas. Indeed, NASA’s Software Working Group is developing mappings between the NASA Software Engineering Requirements, NPR 7150.2, and select industry standards. A mapping to NASA’s Software Assurance Standard exists, and (at the time of writing) mappings to the Software Engineering Institute’s Capability Maturity Model Integration® (CMMI®), and to the Institute of Electrical and Electronics Engineers standard IEEE 12207, are “under review.”

1.3 V&V for Spacecraft Fault Protection

Fault Protection (FP) software on existing NASA robotic spacecraft is a special case of ISHM. In general, ISHM goes beyond such FP in two major aspects: the need for reasoning, primarily as a consequence of the state-space explosion, and, in many applications, the focus on maintaining capability rather than the simpler task of averting catastrophe. Nevertheless, it is worth first considering how V&V is performed for FP before turning attention to ISHM in general.

Ideally, the development process of a spacecraft's FP starts with a detailed fault tree and Failure Modes and Effects Criticality Analysis (FMECA) effort that produces a ”fault set". A fault set is the list of faults that the spacecraft or system might encounter; the fault set can then be subdivided into a "protected fault set" (those for which FP is to be responsible for diagnosing and responding to) and an "unprotected fault set." (those for which FP is not responsible). In order to allocate faults between these two sets, a clear definition of the project's fault tolerance is needed - is it to be single or dual fault tolerant? is the requirement to be fault tolerant or failure tolerant? etc. Having this fault set early in the life of the mission allows for design and risk trade offs as the hardware is selected. It also provides a basis for the amount of redundancy selected for the hardware. Once the protected fault set is determined the fault injection requirements can be specified for the ground support equipment to be used to test the hardware and software.

This is the ideal approach – however, in practice, this rarely occurs in its ideal form. As helpful as it would be to have the full fault set early in the mission, the project often does not have resources to dedicate systems engineers to a thorough fault tree and FMECA effort in early design. Usually, one gets *either* a fault tree *or* a FMECA *drafted*. This means that in practice there is an initial fault set but it is often very partial. The same is true of the fault injection requirements, which, in practice, will be only a partial set in the initial stages.

The best way to overcome these departures from the ideal is to ensure that both fault set development and fault injection requirements identification are on-going processes with milestones at PDR, CDR and individual FP reviews so that the process can be kept somewhat current.

Finally, the FP testing process is itself constrained by project priorities. There is a theoretical desire to begin FP testing early and have it stay in step with the other software and hardware testing. However, in practice the FP testing starts out with low priority, increasing as the overall testing program matures. Logic dictates that in a prioritized environment, there is no need for fault protection testing until the core nominal hardware and software is working. As the testing progresses and confidence in the nominal system matures, then attention turns to the off-nominal cases in which FP plays a central role.

FP testing has the same three levels of V&V as the other areas. It begins with verifying the basic functionality of the fault protection software itself, that is, the fault protection governing software and the monitors and responses. One of the detailed methods used to accomplish this is to “enable” the monitors as soon as possible after a flight software delivery to ensure maximum testing time for detecting errors. The remediation functions are exercised later in the test process as they become available. This testing can range from basic fault testing to a more extreme “stress testing” that involves cascading faults, envelope testing and heavy concurrent load testing. The stress testing completes the triple of verifying requirements, validating capabilities and then stress testing to find out where the system truly fails.

1.4 Example of industry V&V current practice: Space Shuttle Main Engine Controller

The Block II Space Shuttle Main Engine (SSME) Hardware Simulation Laboratory II (HSL II) is the facility utilized for the verification of the SSME Controller (SSMEC) test and flight software. The SSMEC software is used at Stennis Space Center (SSC) for engine checkout and to conduct hot-fire certification tests of the SSME, at Johnson Space Center (JSC) Shuttle Avionics Integration Laboratory (SAIL) supporting the Shuttle Integration Testing, at Kennedy Space Center (KSC) to checkout the SSME in the engine shop, and at KSC to control the SSME during the launch of the Shuttle. The flow for SSMEC software development is shown in Figure 7. The HSL is also used for Avionics integration of hardware prior to installation on the SSME and for Avionics hardware anomaly resolution.

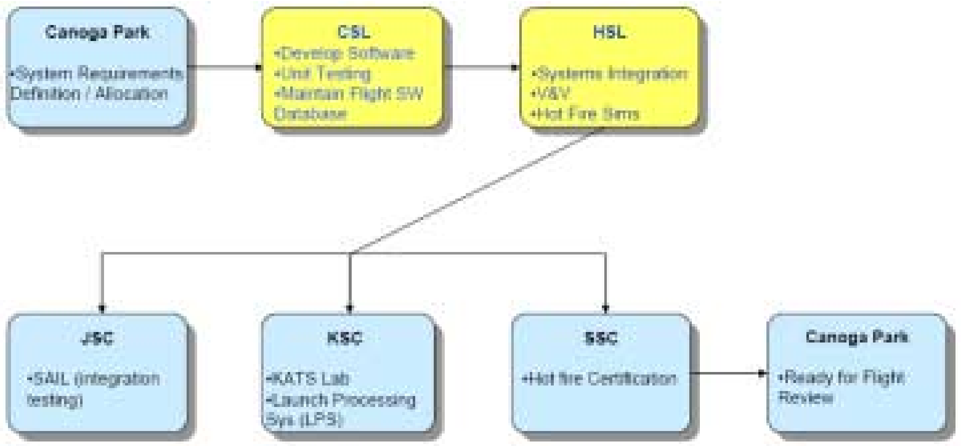

Figure 7. Space Shuttle Main Engine Controller software development flow.

The HSL is an automated verification facility. Laboratory software was developed to accomplish automated testing, digital and analog fault insertion, data collection and analysis, and laboratory calibration.

SSMEC software changes are generated using Requirement Change Notices (RCNs), Design Change Notices (DCNs) and Code Change Notices (CCNs), as appropriate. SSMEC software verification is conducted in the HSL II at MSFC and software certification is conducted on the engine hot-fire test stand at SSC. Changes (RCN/DCN/CCN) are delivered to Rocketdyne HSL personnel at MSFC, who review the changes. Test procedures are generated and/or modified to verify the new requirements or design changes. An executable image compare is performed following each compilation. This compare, against a known base, is used to verify that only the intended software modules were affected and to assist in identifying areas of retest. Each change is then verified in the HSL II. All discrepancies found during the verification process are reported on an SN. Complete, post verification change packages are provided to the SSMEC software community. Rocketdyne at CSL prepares a Hotfire Simulation Request Package that specifies the software configuration, test profile, and special tests, as required. The hotfire simulation and special tests are performed at the HSL II.

In addition, a database compare is performed on the software that is to be used for engine hotfire test. Upon completion of these tests and approval by MSFC, the software is authorized for engine hotfire test at SSC. Engine hotfire tests certify the SSMEC software. Upon completion of the software certification and approval of the ECP and the associated Verification Complete Package by MSFC, the software is then acceptable for STS flight. [Fiorucci et al, 2000]

2 FEASIBILITY AND SUFFICIENCY OF EXISTING SOFTWARE V&V PRACTICES FOR ISHM

In this section we consider whether the existing software development practices can be feasibly applied as-is to ISHM systems, and whether those practices will provide sufficient levels of confidence in ISHM systems.

2.1 Feasibility

NASA's Human-Rating Certification process is defined in NPR 8705.2A (effective date: 2/7/2005). The objective of the human-rating certification process is to document that the critical engineering requirements, health requirements, and safety requirements have been met for a space system that provides "maximum reasonable assurance" that the system's failure will not result in a crew or passenger fatality or permanent disability. This NPR covers numerous aspects of certification, including certification of software. One of the software aspects covered is testing, where one requirement is:

> *1.6.7.1 The Program Manager shall perform testing to verify and validate the performance, security, and reliability of all critical software across the entire performance envelope (or flight envelope) including mission functions, modes, and transitions*

ISHM clearly contains "critical software" and hence is subject to this testing requirement. However, the very nature of ISHM poses significant challenges to meeting this requirement, above and beyond challenges shared by most forms of mission-critical software. Specifically, ISHM, by definition, deals with *off*-nominal conditions in each of its roles (it must recognize, diagnose and respond to: early indications of impending failure, the presence of performance degradations, and failures that have occurred). Several V&V challenges stem from this: it is hard to know that all the significant possible failure modes have been identified (especially for relatively novel components and for conventional components operating in novel conditions); for any given failure mode, its characteristics may not be well understood; there is a large number of ways in which off-nominal conditions can arise (consider all the parts that could fail, and the varying implications of such failure depending on when in the mission it occurs), and the combinations of such failures are vastly more numerous. For example, if there are 1,000 individual possible failures, then there are potentially 1,000,000 *pairs* of such failures (while not every pair will be possible, nevertheless, the number of feasible pairs of failures will tend towards the square of the number of individual failures). This has specific relevance to the feasibility of meeting fault tolerance requirements that may be applicable. For example, another Human Rating requirement states:

> *3.1.1 Space systems shall be designed so that no two failures result in crew or passenger fatality or permanent disability.*

In more general terms, the challenges posed by ISHM systems are that it is hard to assure completeness of models of failure, it is hard to assure that those models are correct, and it is hard to test/inspect/review the very many failure scenarios. While any given failure scenario may itself have a very low likelihood of occurrence, ISHM must be prepared to deal correctly with whichever ones do manifest themselves in the course of the mission, so V&V must address a large fraction of these to achieve the levels of assurance required.

In response to these questions of feasibility, the response could be to evolve the requirements, standards, etc. accordingly, or to leave them as-is and instead rely on provisions for exceptions[1], deviations[2] and waivers[3] from these requirements. In practice waivers have been common. However, since they contradict the intent and effect of requirements, and introduce inconsistencies in the certification process, it is preferable to recognize early-on which requirements cannot be met, and revise these requirements as necessary to preclude reliance upon waivers. The lack of feasibility for "complete" V&V of ISHM under the two-fault design requirement invokes a need to redefine the V&V test requirements. One could establish a V&V testing "floor" in which every fault symptom is simulated in the full flight software environment and the ISHM response is verified.

2.2 Sufficiency

Another question to ask of the existing standards is whether they are sufficient to achieve the levels of assurance desired of ISHM systems.

We begin by noting that even the most stringent of the structural testing levels – the Modified Condition Decision Coverage (MCDC), cannot fully test a realistic software application. To do so would require "path" coverage – testing of every unique sequence of execution through the code. Path coverage is not guaranteed by MCDC. In MCDC each condition is tested largely independently of other decisions in the program, and in a program with *n* binary decision points there are 2^n independent decisions, each of which defines a possible path through the program. Of these, the number that are "feasible" (that is, that can actually be executed by some combination of input data values) is also on the order of 2^n. Thus only a relatively small portion of the possible execution paths are tested even under MCDC. For event-driven (reactive) systems the situation is even worse. ISHM systems fall squarely into this category. As described in the previous subsection, the number of possible behaviors can be a huge number, and the small proportion covered by MCDC would leave most untested.

[1] An *exception* to a requirement can be provided if that requirement is not applicable to every component of the system.

[2] A *deviation* from a requirement can be provided if the requirement cannot be met but there is an alternative method of reducing system risk to an "equivalent or lower" level.

[3] A *waiver* of a requirement may be requested if the requirement is unsatisfied and there is therefore an increased risk.

Further challenges stem from the unusual structure of ISHM software as compared to the more traditional forms of spacecraft software for which the standards, etc., were crafted. ISHM software often makes use of Artificial Intelligence techniques, and is architected accordingly. Specifically, such software typically has both a large, complex "reasoning engine", and "*models*" (e.g., a model might describe the operating modes of the telecommunications system) over which that reasoning engine operates.

The implications for V&V are several:

- conventional approaches to certification, such as measures of code coverage used to gauge the thoroughness of testing, do not take into account those models. In conventional terms, the models would look like data, and typical code coverage metrics would fail to capture the need for coverage of not only the reasoning engine's code, but also the data encoded within the models.
- the overall ISHM system's behavior might be sensitive to small changes in either of the reasoning engine itself (e.g., a small change to a heuristic might lead to drastic changes in performance) or the models (a small change to a model might push the reasoning engine into previously unexplored regimes of behavior) – it is hard to predict (and therefore hard to be sure to have adequately exercised with testing) when and how these small changes will affect ISHM behavior
- the performance (run time, memory consumption, CPU utilization) of reasoning engines themselves, because of their heuristic nature, is hard to guarantee. If they are operating close to the computational "cliff" (where performance degrades rapidly as the problem complexity increases only slightly), they will exhibit occasional wild fluctuations from "normal" – for many runs it may perform within expected bounds, but once in a while, the performance is extremely poor (slow, huge memory usage, …).

ISHM must correctly report failure conditions, and, importantly, must avoid "false alarms". Both of these require that ISHM take as input uncertain data, and yield information and decisions with high(er) certainty. For example, ISHM needs to distinguish engine failure from failure of the sensor(s) monitoring the engine's health (those sensors are fallible devices, and may themselves fail; in fact, sensors are generally considered less reliable than the components they are monitoring). The ISHM algorithms (and implementation thereof) that perform its certainty-increasing process must be extremely reliable, since they will be in continuous operation.

Lastly, many of the systems whose health ISHM is to manage will themselves contain software. In such cases ISHM may be expected to be cognizant of, and responsive to, the health of those systems' software. However, software "failure" does not completely parallel hardware "failure" (software doesn't "wear out", rather, during operation a latent defect – "bug" – in the software may become manifest in the particular execution path it follows). Therefore it is less well understood whether ISHM techniques can accommodate failure modes that have their origin in latent software defects (predict them for prognosis purposes, diagnose them once they have occurred, and in either case know what to do in response). There are approaches to containing faults within the software systems itself: traditional exception handling is code to trap and respond appropriately to faults, e.g., divide-by-zero; N-version programming [Avizienis & Chen, 1977] provides software redundancy, by comparing the results returned by N programs that have been *independently* developed from the same specification. For detection of faults that evade such containment, runtime software-fault monitoring (for a recent survey, see [Delgado et al, 2004] is an approach in which the software's execution-time behavior is compared to specified properties; non-compliance with one or more of those properties would be an input to ISHM.

3 OPPORTUNITIES FOR EMERGING V&V TECHNIQUES SUITED TO ISHM

The unusual nature of ISHM software raises both challenges for V&V and certification (outlined in the previous section), and opportunities to amplify the efficacy of existing techniques, and to make use of some new and emerging V&V techniques that offer the promise of overcoming some of those key challenges. This section describes the origins of those opportunities, and gives some representative examples of emerging V&V techniques.

3.1 ISHM architecture

Emerging forms of ISHM are likely to be architected using a combination of hierarchical composition (with each subsystem performing its own health management, but propagating its status, and if necessary the faults it cannot manage locally, to the system of which it is a part, and so on), and model-based reasoning where a generic reasoning engine operates over system-specific models.

Hierarchical composition potentially favors V&V by allowing analysis itself to take advantage of the hierarchy, subdividing the V&V into manageable portions. V&V of this kind, often referred to as “hierarchical verification” or “compositional verification”, is an area of current interest within the V&V community. For a discussion of some of the issues, see [Martin & Shukla, 2003]; for an example of a whole workshop focused on the topic, see [de Boer & Bonsangue, 2004] . Some of this work has been applied to NASA missions, e.g., [Giannakopoulou & Penix, 2004].

Model-based approaches to ISHM yield an ISHM system architecture divided into a generic, and therefore reusable, reasoning engine, and system-specific models. The reasoning engine itself is a non-trivial piece of software, and so the correctness of its implementation needs to be checked. However, since it will be reused from application to application, the effort it takes to check that implementation can be amortized over those multiple applications.

Whatever the architecture, V&V of ISHM will require assuring the correctness of its core algorithms (e.g., voting schemes); this kind of problem has long been appropriate for formal methods such as theorem proving (e.g., [Rushby 1991]). Also, ISHM systems may be expected to be amenable to traditional software reliability engineering techniques based on measurements of defect discovery and removal during development and test: see [Musa, 1998], [Vouck, 2000] for overviews of this field. Methods that can expand the information gained from individual test cases would be useful for testing of the numerous behaviors that ISHM systems can exhibit – an example of such a method is the recognition of inconsistent uses of shared variables in a test run, even if no classical race condition occurs within that run [Artho et al, 2003].

3.2 Models used in ISHM

In order that ISHM can perform its reasoning (e.g., diagnose the cause of a fault from a set of symptoms), those models are designed to be machine-manipulable, by the ISHM reasoning engine itself. V&V can also benefit from such machine-manipulable models. As stated in [Menzies & Pecheur, 2005], "*These models are often* declarative *and V&V analysts can exploit such declarative knowledge for their analysis*".

Many of the emerging V&V techniques perform analysis – for V&V purposes – over the same kinds of models that ISHM utilizes. The adoption of those V&V techniques in *traditional* software settings has always been impeded by the need to construct such models by hand, from the various forms of system documentation intended for human, but not computer, perusal (e.g., requirements stated in paragraphs of English). This has made them costly and time-consuming to use, and as a result their application has, in practice, been limited to only the most critical core elements of software and system designs (for an in-depth discussion, see [Rushby, 1993]). A representative example drawn from the spacecraft fault protection domain is [Schneider et al, 1998]'s use of "model checking" applied to the checkpoint and rollback scheme of a dually redundant spacecraft controller. In contrast, in model-based ISHM, such models are available early in the lifecycle, the ideal time to benefit from the results of analysis. Automatic translation from the form of ISHM-like models to the form of V&V models has been shown to be feasible, e.g., [Pingree et al, 2002] illustrates such an approach in which they translate statecharts into the input form for the model checker SPIN; [Pecheur & Simmons, 2000] translate models in Livingstone (a model-based health management system [Williams & Nayak, 1996]) into the model checker SMV. [Penix et al., 1998] reports experiments to translate AI planner domain models into SMV, SPIN and Murphi model checkers, allowing a comparison of how the different systems would support specific types of validation tasks.

Traditional techniques such as testing can also leverage the availability of such models. For example, [Blackburn et al, 2002] describes test automation (generation of the test cases, test drivers, and test result evaluation) utilizing models, demonstrated on the ill-fated Mars Polar Lander software design. Human-conducted activities such as reviews and inspections may be well-suited to scrutiny of declarative models.

Another source of opportunity offered by model-based reasoning is that the reasoning software can yield both its result (e.g., a diagnosis), and the chain of reasoning that led to that result. That chain of reasoning provides opportunities for cross-checking – not only checking that the result is correct, but also that it is correct for the right reasons (e.g., all the appropriate information was taken into account when arriving at its conclusion). For an example of this used during testing of an AI planner, see [Feather & Smith, 2001].

An important property of ISHM systems is that they are adequate to support diagnosis of a specified class of faults. Often termed *diagnosability*, this means that using the information available from sensors, the ISHM system is always able to distinguish whenever the system is in a fault state, and if so, disambiguate which fault state it is. Note that this is a property of a combination of the system itself (what states it can exhibit), the sensors (what information about the system state they make available to ISHM), and the reasoning capabilities of the ISHM system itself. For example, if among the system's possible behaviors there are two scenarios that lead to system states required to be distinguished, and yet the sensor information made available to the ISHM system is exactly the same for both those scenarios, then it would clearly be impossible for the ISHM system to make the distinction. For a discussion of diagnosability and approaches to its attainment, see [Sampath et al., 1995] and [Jiang et al., 2002]. An approach to verification of this property is described in [Cimatti et al, 2003].

For V&V of the system as a whole, [Lindsey & Pecheur, 2003] and [Lindsey & Pecheur, 2004] discuss an approach that focuses on advanced simulation of the actual software (as opposed to verification of the model only). Concretely, this has been implemented in the Livingstone PathFinder (and Titan PathFinder) framework. Although this approach does not address diagnosability directly, it can catch diagnosis errors that may be traced back to diagnosability issues. They discuss an application of this approach to the main propulsion feed subsystem of the X-34 space vehicle.

3.3 Planning systems in ISHM

In addition to diagnosing the health status of the systems they monitor, many ISHM systems will be required to plan the appropriate actions to recover from unhealthy states, and to execute those actions. Model-based techniques will play an increasingly prominent role in the planning and execution stages, just as in the diagnosis. Artificial Intelligence techniques for response planning have the same reasoning engine + models architecture, and so are prone to the same V&V challenges and opportunities as diagnosis systems. In addition, a plan execution system ("executive") is needed to execute the plans. V&V of this software system must ensure that the execution of the commands and the response of the fault protection system conforms to pre-planned behavior. [Varma et al., 2005] discusses an executive built with plan verifiability in mind. [Brat et al, 2003] describes the results of applying several verification tools to an executive for a robotic Martian rover.

3.4 ISHM of software systems

Advances in the understanding of faults in software systems will be applicable when, as is very likely, ISHM has within its scope the management of systems with significant reliance on software.

Risk analysis methods that serve to identify software vulnerabilities have been adapted for software systems – *Software* Failure Modes Effects and Criticality Analysis (SFEMCA) [Hall et al, 1983], and *Software* Fault Tree Analysis (SFTA) [Leveson, 1995]. Ongoing work in this area includes means to combine these approaches [Lutz & Woodhouse, 1999], and to apply quantitative techniques adapted from Probabilistic Risk Assessment (PRA) to software [Li et al, 2003], [Feather 2004].

Detection of software faults during operation will be a key element of ISHM. The field of "runtime software-fault monitoring" is surveyed in [Delgado et al, 2004]; for an application to fault protection on a space system, see [Drusinsky & Watney, 2003].

4 V&V FOR CONSIDERATIONS FOR ISHM SENSORS AND AVIONICS

Integrated System Health Management (ISHM) relies on information derived from sensors by avionics (signal conditioning, data conversion and data processing hardware) to assess the state of the system. The performance of the ISHM system is dependent upon the fault coverage by the sensors embedded in the space vehicle. The quality of data from the sensors and the overall reliability of the hardware of the ISHM system are critical to ISHM performance. In addition to meeting functional requirements, the ISHM system must be certified to operate reliably in the space environment. Environmental requirements for certification will include launch vehicle dynamics (vibration, shock and acoustic), thermal environments (orbital and re-entry), electromagnetic compatibility, and radiation (primarily single-event effects, since man-rated vehicles typically minimize exposure to trapped radiation environments).

4.1 ISHM Hardware Certification

Spaceflight hardware certification ensures that the ISHM hardware meets specified design, manufacturing, life, and environmental requirements. For new hardware, preliminary and critical design reviews are conducted to ensure that all basic safety, reliability, and quality assurance requirements are met. A certification requirements document identifies and defines the induced and natural environments and methodologies used in the certification approach. Certification consists of qualification testing and analysis at the highest practical level of assembly, installation, or system. Qualification testing must consider functional, environmental, and life requirements. Certification may be achieved by analysis when testing is not necessary, feasible, or cost-effective.

4.2 Spaceflight Hardware V&V

Spaceflight hardware is generally developed via a requirements-driven process where the capabilities, performance specifications and physical characteristics are developed within the constraints of mission resource allocations. High-level (system) requirements are translated into lower-level requirements, ultimately resulting in specifications that become the basis for hardware design. Validation is performed via thorough requirements traces (upward and downward) to ensure correct requirements are established at all levels. Throughout the hardware development, the compliance of the hardware design with the requirements is verified early in design reviews and later, in the hardware test program. Often a matrix is generated and maintained to track the verification of the hardware against requirements on that hardware. A performance baseline for verification of hardware functionality is established prior to subjecting the hardware to a battery of environmental tests. Abbreviated functional testing is frequently performed during the series of environmental tests (i.e. between vibration tests on each axis of the hardware). Testing of payload or subsystem avionics hardware is generally performed at the electronics box level prior to delivery to the space vehicle for integration.

System-level functional testing is often performed with engineering model or prototype subsystem hardware early in the integration phase. Testbeds are frequently employed to develop system-level functionality (command and data handling subsystems). Flight hardware can be verified in testbeds that have the appropriate interfaces and hardware protection. During integration of the space vehicle, flight hardware subsystems are typically connected to the space vehicle power and data systems via a “safe-to-mate” verification procedure. Pin-level verification of the interfaces are performed through “break-out box” equipment until the unit being integrated has been powered and proper communication is verified. Only then is the unit directly mated to the flight system connectors. After all of the flight hardware has been integrated, system-level testing is completed. Robotic spacecraft typically undergo system-level environmental testing (EMI/EMC, vibration, acoustic, system thermal-vacuum tests) to verify system performance in simulated launch and space environments.

4.3 Sensor Data V&V

Due to the potentially large number of sensors, many of which are exposed to harsh environments, the ISHM system must be tolerant of sensor faults. The processes for the selection, qualification and installation of sensors are important factors for minimizing sensor faults. An ISHM system should be able to validate sensor readings and diagnose sensor faults in real-time. The area of Sensor Failure Detection, Isolation and Accommodation (SFDIA) is being addressed by two conceptually different approaches: physical and analytical redundancy.

4.3.1 Physical redundancy

Traditional flight control systems deploy triple or quadruple physical redundancy in their network of sensors to achieve the level of reliability necessary for manned spacecraft or aircraft certification. Physical redundancy SFDIA techniques are based on voting and mid-value selection schemes. It is clear that there are penalties such as mass, power, volume, and cost associated with a physical redundancy approach to the SFDIA problem.

4.3.2 Analytical redundancy

Most of the current research activities on SFDIA focus on the use of analytical redundancy techniques. A partial list of analytical SFDIA techniques includes Generalized Likelihood Ratio (GLR); Multiple Model, Extended, and Iterative Extended Kalman Filtering (MMKF, EKF and IEKF); Sequential Probability Likelihood Ratio Test (SPLRT), and Generalized Likelihood Test/Maximum Likelihood Detector (GLT/MLD).These techniques feature a continuous monitoring of the measurements from the sensors. At nominal conditions, these signals follow some known patterns with a certain degree of uncertainty due to the presence of system and measurement noise. However, when sensor failure occurs, the observable outputs deviate from the predicted values calculated on-line or off-line from an estimation scheme generating a residual. A sensor failure can be declared when the associated residual exceeds, for a single or for multiple time instants, a certain numerical threshold.

Analytical redundancy and Bayesian decision theory were combined to produce a sensor validation system concept for real-time monitoring of Space Shuttle Main Engine telemetry (see [Bickford et al, 1999]).The validation system, as illustrated in the block diagram below (Figure 2), was implemented in Ada and hosted on a Boeing X-33 prototype flight computer (R3000 at 25 MHz). SSME telemetry was played back at real-time rate through the system at the Marshall Avionics System Testbed (MAST). Data from 50 SSME flight firings were processed at real-time rates and 3 sensor failures were correctly identified.

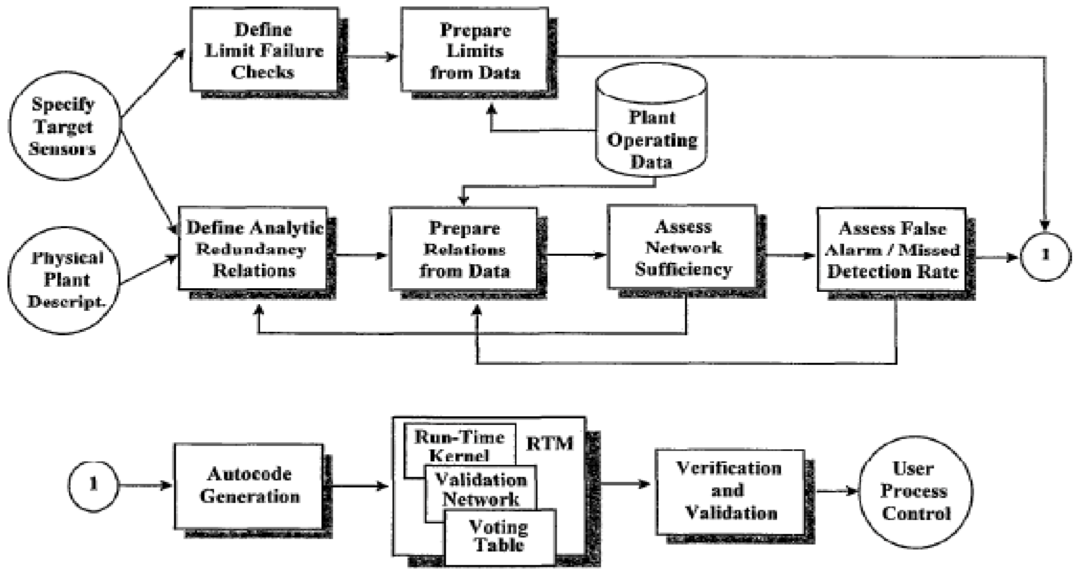
Define Limit Failure Checks
Prepare Limits from Data
Specify Target Sensors
Plant Operating Data
Define Analytic Redundancy Relations
Prepare Relations from Data
Assess Network Sufficiency
Assess False Alarm / Missed Detection Rate
1
Physical Plant Descript.
1
Autocode Generation
Run-Time Kernel
RTM
Validation Network
Voting Table
Verification and Validation
User Process Control

Figure 2. Block diagram for the real-time SSME sensor data validation concept developed by Bickford, et al.

More recently, neural network (NN) approaches to sensor data validation have been developed. As an example, data from a Boeing 737 was processed via a NN-based on-line learning scheme [Napolitano et al, 1999]. The Extended Back Propagation (EBP) algorithm was used by the authors for the on-line learning. The algorithm was selected for its performance in terms of learning speed, convergence time, and stability when compared to the conventional Back Propagation (BP) algorithm. The SFDIA scheme is illustrated in the block diagram shown in Figure 2. It consists of a main NN (MNN) and a set of ‘n’ decentralized NNs (DNNs), where ‘n’ is the number of the sensors in the flight control system for which a SFDIA is desired. The outputs of the MNN replicate, through on-line prediction, the actual measurements from the ‘n’ sensors with one time instant delay, that is a prediction of the state at time ‘k’ using measurements from ‘k-l’ to ‘k-p’ to be compared with the actual measurement at time ‘k’. In their study, the authors processed flight data obtained from about 10,000 seconds of B737 flight recorder data to train the MNN and DNNs. Simulated sensor failures were injected to test the response of the NN. They were able to demonstrate rapid on-line learning and proper identification of a variety of sensor failures both hard (complete sensor signal loss) and soft (drift) and to have the failed sensor data accommodated by the physical model adapted by the on-line learning process.

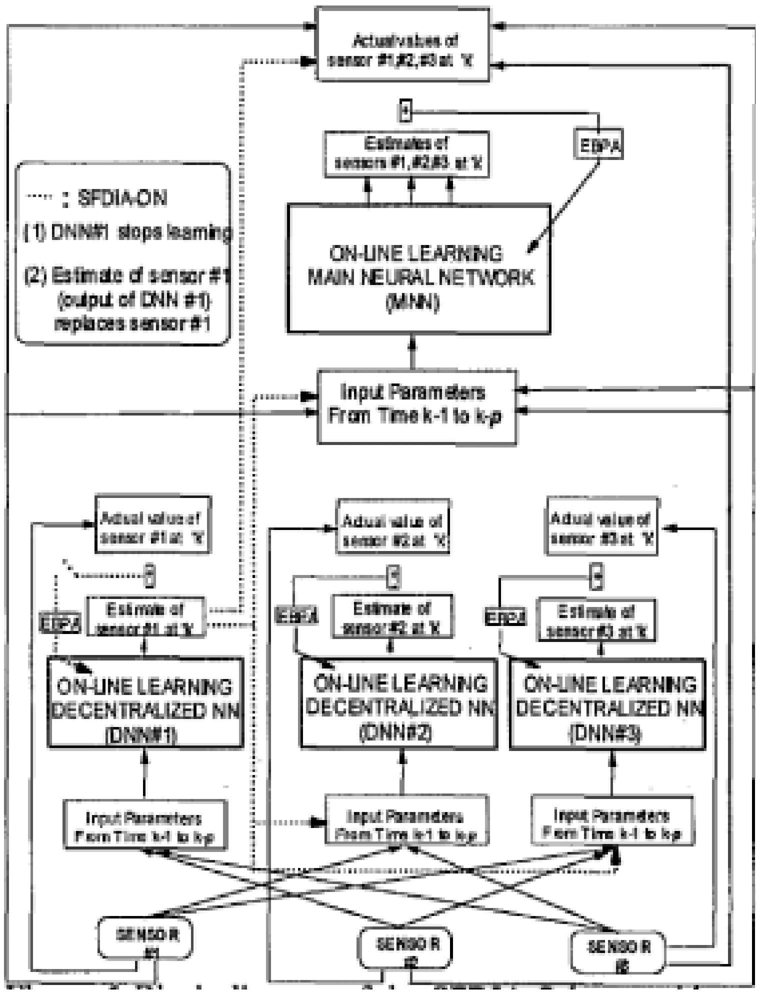
Actual values of sensor #1,#2,#3 at 'k'
Estimates of sensors #1,#2,#3 at 'k'
EBPA
ON-LINE LEARNING MAIN NEURAL NETWORK (MNN)
Input Parameters From Time k-1 to k-p
: SFDIA-ON
(1) DNN#1 stops learning
(2) Estimate of sensor #1 (output of DNN #1) replaces sensor #1
Actual value of sensor #1 at 'k'
Actual value of sensor #2 at 'k'
Actual value of sensor #3 at 'k'
Estimate of sensor #1 at 'k'
Estimate of sensor #2 at 'k'
Estimate of sensor #3 at 'k'
ON-LINE LEARNING DECENTRALIZED NN (DNN#1)
ON-LINE LEARNING DECENTRALIZED NN (DNN#2)
ON-LINE LEARNING DECENTRALIZED NN (DNN#3)
SENSOR #1
SENSOR #2
SENSOR #3

Figure 2. Block diagram of NN SFDIA scheme showing result of a failure in Sensor #1.

5 SUMMARY

It is apparent that a combination of *multiple* V&V approaches are required for ISHM. For example, depending on the ISHM architecture, traditional testing approaches may be adequate and appropriate for some layers of ISHM functionality, whereas the use of AI techniques such as reasoning under uncertainty and mission planning (and re-planning) have characteristics that set them apart and challenge standard V&V techniques. Most notably, AI techniques based on explicit use of model-based reasoning exhibit algorithmic and implementation intricacies (within their AI reasoners themselves) on a par with other complex software systems, but in addition the behaviors they may exhibit during operation depend critically on the models themselves (elements that traditional V&V has not had to deal with). Fortuitously, the additional V&V challenges their model-based nature gives rise to are balanced by the enhanced opportunities to apply certain V&V techniques, especially those based on analytic methods.

The function of ISHM is to increase the reliability of the system whose health it manages. To do this, ISHM must, in the vast majority of cases, correctly ascertain the status of the system, despite the fallibility of both the system itself and also the sensors that monitor the status of the system. Thus the reliability of the ISHM's core functionality (e.g., its voting algorithms that adjudicate among multiple – some possibly erroneous – readings from multiple sensors) must be among the most reliable software systems on the entire vehicle. These considerations, coupled with the non-traditional architecture that many ISHM systems employ (notably model-based reasoning), call into question both the feasibility and adequacy of existing standards, practices, etc., for V&V and certification if they are to encompass ISHM. Overall, therefore, it is necessary to modify V&V and certification process to permit the use of ISHM, to find the right mix of V&V and certification methods to match the architecture of the vehicle and ISHM system, and to guide the maturation of emerging V&V techniques to support their application to ISHM.

An aspect of ISHM is that it must interact with other control systems, including humans (both ground operations, and on-board crew members). These interactions also raise important concerns for V&V, but are outside the scope of this paper.

6 ACKNOWLEDGMENT

The research described in this paper was carried out by NASA Ames Research Center, NASA Stennis Space Center, and the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

7 REFERENCES

[Artho et al, 2003] Artho, C., Havelund, K. & Biere, A. "High-level data races" *Software Testing, Verification and Reliability* Vol 13, No 4, pp 207-227, Nov 2003.

[Avizienis & Chen, 1977] Avizienis, A. & Chen, L. “On the implementation of N-version programming for software fault tolerance during execution” *Proc. IEEE COMPSAC 77*, pp. 149-155, Nov. 1977.

[Bickford et al, 1999] Bickford, R.L., Bickmore, T.W., Meyer, C.M., Zakrajsek, J.F. “Real-Time Sensor Data Validation for Space Shuttle Main Engine Telemetry Monitoring”, AIAA-1999-2531, *35th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, Los Angeles, California, 20-24 June 1999.

[Blackburn et al, 2002] Blackburn, M., Busser, R., Nauman, A., Knickerbocker, R. & Kasuda, R. “Mars Polar Lander fault identification using model-based testing” *8th IEEE International Conference on Engineering of Complex Computer Systems*, 2-4 Dec. 2002 pp. 163-169.

[Brat et al, 2003] Brat, G., Giannakopoulou, D., Goldberg, A., Havelund, K., Lowry, M., Pasareanu, C.S., Venet, A. & Visser, W. “Experimental Evaluation of Verification and Validation Tools on Martian Rover Software” *Proc. SEI Software Model Checking Workshop*, Formal Methods in System Design, vol. 25, issue 2, Pittsburgh, PA, pp. 167-198, Mar. 24, 2003.

[Cimatti et al, 2003] Cimatti, A., Pecheur, C. & Cavada, R. “Formal Verification of Diagnosability via Symbolic Model Checking” *Proc. IJCAI 2003: 18th Int'l Joint Conf. on Artificial Intelligence*, Acapulco, Mexico, pp. 501-503, Aug. 9-15, 2003
http://ti.arc.nasa.gov/ase/papers/MOCHART02/VVDiag-ECAI.pdf

[de Boer & Bonsangue, 2004] de Boer, F. & Bonsangue, M. (eds), Proceedings of the Workshop on the Verification of UML Models, Oct 2003, *Electronic Notes in Theoretical Computer Science*, Volume 101, pp 1-179, 2004.

[Delgado et al, 2004] Delgado, N., Gates, A.Q. & Roach, S. “A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools” *IEEE Transactions on Software Engineering*, Vol 30, No 12, December 2004, pp 859-872.

[Drusinsky & Watney, 2003] Drusinsky, D. & Watney, G. “Applying Run-Time Monitoring to the Deep-Impact Fault Protection Engine”, *28th Annual NASA Goddard Software Engineering Workshop*, pp 127-133, Dec. 2003.

[Feather & Smith, 2001] Feather, M.S. & Smith, B. “Automatic Generation of Test Oracles – From Pilot Studies to Application”, *Automated Software Engineering* (Kluwer); Vol 8, No 1, Jan 2001, 31-61

[Feather et al, 2001] Feather, M.S., Fickas S. & Razermera-Mamy, N-A. “Model-Checking for Validation of a Fault Protection System”, *IEEE 6th International Symposium on High Assurance Systems Engineering*; Boca Raton, Florida, Oct 2001, pp. 32-41.

[Feather, 2004] Feather, M.S. “Towards a Unified Approach to the Representation of, and Reasoning with, Probabilistic Risk Information about Software and its System Interface” *ISSRE 2004 - 15th IEEE International Symposium on Software Reliability Engineering*; Saint-Malo, Bretagne, France, Nov 2-5 2004;

[Fiorucci et al, 2000] Fiorucci, T.R, Lakin II, D.R., and Reynolds, T.D. “Advanced Engine Health Management Applications of the SSME Real-Time Vibration Monitoring System” *36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, Huntsville, AL, July 16-19, 2000.
http://pdf.aiaa.org/getfile.cfm?urlX=%2D5%26%5D0%3BU%2BDN%26S7SPGNW%20%20%20%0A&urla=%25%2ARH%25%230H%20%0A&urlb=%21%2A%20%20%20%

0A&urlc=%21%2A0%20%20%0A&urld=%26%2BR%2C%20%21%40%22E%0A&urle=%26%2BR%40%20%22%40NC%0A

[Fudge et al, 2003] Fudge, M., Stagliano, T. & Tsiao, S. "Non-Traditional Flight Safety Systems & Integrated Vehicle Health Management Methods: Descriptions of Proposed & Existing Systems and Enabling Technologies & Verification Methods". Final Report for the Office of the Associate Administrator for Commercial Space Transportation, FAA, Section AST-300. August 26, 2003.

[Giannakopoulou & Penix, 2004] Giannakopoulou, D. & Penix, J. "Component Verification and Certification in NASA Missions", *4th ICSE Workshop on Component-Based Software Engineering*, 2004.

[Hall et al, 1983] Hall, F.M., Paul, R.S. & Snow, W.E. "Hardware/Software FMECA", *Proc Annual Reliability and Maintainability Symposium*, 1983, pp. 320-327.

[Jiang et al. 2002] Jiang, S. & Kumar, R. "Failure Diagnosis of Discrete Event Systems with Linear-time Temporal Logic Specifications". *IEEE Transactions on Automatic Control*, Submitted.

[Johnson, 1998] (Schad) Johnson, L. "DO-178B, "Software Considerations in Airborne Systems and Equipment Certification" October 1998, available at: http://www.stsc.hill.af.mil/crosstalk/1998/10/schad.asp

[Leveson, 1995] Leveson, N. "*Safeware, System Safety and Computers*" Addison-Wesley, 1995.

[Li et al, 2003] Bin Li; Ming Li; Ghose, S. & Smidts, C.S. "Integrating software into PRA", *14th International Symposium on Software Reliability Engineering (ISSRE 2003)*. 17-20 Nov. 2003 pp 457 – 467

[Lindsey & Pecheur, 2004] Lindsey, T. & Pecheur, C. "Simulation-Based Verification of Autonomous Controllers with Livingstone PathFinder" *Proc. TACAS'04: Tenth Int'l Conf. on Tools And Algorithms For The Construction And Analysis Of Systems*, Springer LNCS, vol. 2988, Barcelona, Spain, pp. 357-371, Mar. 29 to Apr. 2, 2004. http://ti.arc.nasa.gov/ase/papers/TACAS04/lpf-tacas04.pdf

[Lindsey & Pecheur, 2003] Lindsey, T. & Pecheur, C. "Simulation-Based Verification of Livingstone Applications" *Proc. DSN 2003: Int'l Conf. on Dependable Systems and Networks*, San Francisco, CA, pp. 741-750, June 22-25, 2003. http://ti.arc.nasa.gov/ase/papers/DSN03/lpf-dsn03.pdf

[Lutz & Woodhouse, 1999] Lutz, R.R. and Woodhouse, R.M. "Bi-directional Analysis for Certification of Safety-Critical Software" *1st International Software Assurance Certification Conference (ISACC'99)*, Feb 28-Mar 2, 1999

[Martin & Shukla, 2003] Martin, G. & Shukla, S. "Panel: hierarchical and incremental verification for system level design: challenges and accomplishments", *MEMOCODE '03, Formal methods and models for Co-Design*, pp. 97-99, 2003.

[Menzies & Pecheur, 2005] Menzies, T. and Pecheur, C. "Verification and Validation and Artificial Intelligence". In Zelkowitz, M. (ed.), *Advances in Computers*, Volume 65. Elsevier, 2005.

[Meyer & Zakrajsek, 1993] Meyer, C.M & Zakrajsek, J.F. "Rocket Engine Failure Detection Using System Identification Techniques" NASA Contractor Report 185259, AAIA-90-1993.

[Musa 1998] *Software Reliability Engineering*, McGraw-Hill, New York, 1998.

[Napolitano et al, 1999] Napolitano, M., An, Y., Seanor, B., Pispitsos, S., and Martinelli, D. “Application of a Neural Sensor Validation Scheme to Actual Boeing 737 Flight Data” AIAA-1999-4236, *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Portland, OR, Aug. 9-11, 1999

[Nelson & Pecheur, 2002] Nelson, S. & Pecheur, C. “Formal Verification of a Next-Generation Space Shuttle” *Proc. FAABS 2002: Second Workshop on Formal Aspects of Agent-Based Systems*, Springer LNCS, vol. 2699, Greenbelt, MD, pp. 53-67, Oct. 28-30, 2002. http://ti.arc.nasa.gov/ase/papers/FAABSII/VVofIVHM.pdf

[Pecheur & Simmons, 2000] Pecheur, C. & Simmons, R. “From Livingstone to SMV: Formal Verification for Autonomous Spacecrafts” *1st Goddard Workshop on Formal Approaches to Agent-Based Systems*, pp 5-7, April 2000.

[Penix et al., 1998] Penix, J., Pecheur, C., & Havelund, K. "Using Model Checking to Validate AI Planner Domain Models," *Proc. SEL'98: 23rd Annual Software Engineering Workshop,* NASA Goddard, Dec. 1998.
http://ti.arc.nasa.gov/ase/papers/SEL98/Penix-SEL-Workshop.pdf

[Pingree et al, 2002] Pingree, P.J., Mikk, E., Holzmann, G.J., Smith, M.H. & Dams, D. “Validation of Mission Critical Software Design and Implementation using Model Checking”, *The 21st Digital Avionics Systems Conference*, Volume 1, pp 6A4-1 – 6A4-12, Oct 2002.

[Rushby 1991] Rushby, J. “Formal verification of algorithms for critical systems”, *Proceedings of the conference on Software for Critical Systems*, pp. 1-15,, ACM, 1991.

[Rushby, 1993] Rushby, J. “Formal Methods and the Certification of Critical Systems” *Technical Report CSL-93-7*, Dec 1993, SRI International, Menlo Park, CA.

[Sampath et al., 1995] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D.C. “Diagnosability of discrete-event systems”, *IEEE Trans. Automat. Control* 40(9) (1995) 1555-1575.

[Schneider et al, 1998] Schneider, F., Easterbrook, S.M., Callahan, J.R. & Holzmann, G.J. “Validating requirements for fault tolerant systems using model checking”, *3rd Int. Conf. on Requirements Engineering*, 6-10 Apr 1998, pp 4-13.
http://gltrs.grc.nasa.gov/cgi-bin/GLTRS/browse.pl?1990/CR-185259.html

[Tumer & Bajwa, 1999] Tumer, I. & Bajwa, A. “A survey of aircraft engine health monitoring systems” AIAA-99-2528. *35th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, Los Angeles, CA., June 20-23, 1999.
ic.arc.nasa.gov/publications/pdf/1998-0032.pdf

[Vouk, 2000] Vouk, M.A., “Software reliability engineering”, *Proceedings of the 2000 Annual Reliability and Maintainability Symposium (RAMS),* Los Angeles, CA, IEEE Computer Society.

[Varma et al., 2005] Verma, V., Estlin, T., Jonsson, A., Pasareanu, C., & Simmons, R. “Plan Execution Interchange Language (PLEXIL) for Command Execution”, International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS), 2005.
http://ti.arc.nasa.gov/people/pcorina/papers/vandi_isairas05.pdf

[Williams & Nayak, 1996] Williams, B.C. & Nayak, P.P., “A Model-based Approach to Reactive Self-Configuring Systems”, *Proceedings of AAAI-96*, 1996.